Kernel Korner: The ELF Object File Format by Dissection

Date: Monday, May 01, 1995
Topic: Linux Kernel

Eric Youngdale

The Executable and Linking Format has been a popular topic lately, as people ask why the kernel configuration script asks whether or not to configure loading ELF executables. Since ELF will eventually be the common object file format for Linux binaries, it is appropriate to document it a bit. Last month, Eric introduced us to ELF, and this month, he gives us a guided tour of real ELF files.

Last month, we reached a point where were beginning to dissect some real ELF files. For this, I will use the **readelf** utility which I wrote when I was first trying to understand the ELF format itself. Later on, it became a valuable tool for debugging the linker as I added support for ELF. The sources to readelf should be on tsx-11.mit.edu in pub/linux/packages/GCC/src or in pub/linux/BETA/ibcs2.

Let us start with a very simple program--the hello world program we used last month.

```
largo% cat hello.c
main()
{
        printf("Hello World
");
}
largo% gcc-elf -c hello.c
```

On my laptop, the gcc-elf command invokes the ELF version of gcc--once ELF becomes the default format, you will be able to use the regular gcc command which produces the ELF file hello.o. Each ELF file starts with a header (`struct elfhdr` in /usr/include/linux/elf.h), and the readelf utility can display the contents of all of the fields:

```
largo% readelf -h hello.o
ELF magic:7f 45 4c 46 01 01 01 00 00 00 00 00
        00 00 00 00
Type, machine, version = 1 3 1
Entry, phoff, shoff, flags = 0 0 440 0
ehsize, phentsize, phnum = 52 0 0
shentsize, shnum, shstrndx = 40 11 8
```

The ELF `magic` field is just a way of unambiguously identifying this as an ELF file. If a file does not contain those 16 bytes in the `magic` field, it is not an ELF file. The type, machine, and version fields identify this as an ET_REL file (i.e., an object file) for the i386. The `ehsize` field is just the `sizeof(struct elfhdr)`.

Each ELF file contains a table that describes the sections within the file. The `shnum` field indicates that there are 11 sections; the `shoff` field indicates that the section header table starts at byte offset 440 within the file. The `shentsize` field indicates that the entry for each section is 40 bytes long. All throughout ELF, the sizes of various structures are always explicitly stated. This allows for flexibility; the structures can be expanded as required for some hardware platforms and the standard ELF tools do not have to know about this to be able to make sense of the binary. Also, it allows room for future expansion of the structures by newer versions of the standard.

```
largo% readelf -S hello.o
There are 11 section headers, starting at offset 1b8:
[0]              NULL           00000000 00000 00000 00 / 0 0 0 0
[1] .text        PROGBITS       00000000 00040 00014 00 / 6 0 0 10
[2] .rel.text    REL            00000000 00370 00010 08 / 0 9 1 4
[3] .data        PROGBITS       00000000 00054 00000 00 / 3 0 0 4
[4] .bss         NOBITS         00000000 00054 00000 00 / 3 0 0 4
[5] .note        NOTE           00000000 00054 00014 00 / 0 0 0 1
[6] .rodata      PROGBITS       00000000 00068 0000d 00 / 2 0 0 1
[7] .comment     PROGBITS       00000000 00075 00012 00 / 0 0 0 1
[8] .shstrtab    STRTAB         00000000 00087 0004d 00 / 0 0 0 1
[9] .symtab      SYMTAB         00000000 000d4 000c0 10 / 0 a a 4
[a] .strtab      STRTAB         00000000 00194 00024 00 / 0 0 0 1
```

Listing 1. Section Table for hello.o

Each section header is just a `struct ELF32_Shdr`. You may notice that the name field is just a number--this is not a pointer, but an offset into the .shstrtab section (we can find the index of the .shstrtab section from the file header in the `shstrndx` field). Thus we find the name of each section at the specified offset within the .shstrtab section. Let us dump the section table for this file; see figure 1. You will notice sections for nearly everything which we have already discussed. Each section has an identifier which specifies what the section contains (in general, you should never have to actually know the name of a section or compare it to anything).

After the type, there is a series of numbers. The first of these is the address in virtual memory where this section should be loaded. Since this is a .o file, it is not intended to be loaded into virtual memory, and this field is not filled in. Next is the offset within the file of the section, and then is the size of the section. After this come a series of numbers--I won't parse these in detail for you, but they contain things like the required alignment of the section, a set of flags which indicate whether the section is read-only, writable, and/or executable.

The readelf program is capable of performing disassembly:

```
largo% readelf -i 1 hello.o
0x00000000  pushl      %ebp
0x00000001  movl       %esp,%ebp
0x00000003  pushl      $0x0
0x00000008  call       0x08007559
0x0000000d  addl       $4,%esp
0x00000010  movl       %ebp,%esp
0x00000012  popl       %ebp
0x00000013  ret
```

The .rel.text section contains the relocations for the .text section of the file, and we can display them as follows:

```
largo% readelf -r hello.o
Relocation section data:.rel.text (0x2 entries)
Tag: 00004 Value 00301 R_386_32    (0 )
Tag: 00009 Value 00b02 R_386_PC32  (0 printf)
```

This indicates that the .text section has two relocations. As expected, there is a relocation for printf indicating that we must patch the address of printf into offset 9 from the beginning of the .text section, which happens to be the operand of the call instruction. There is also a relocation so that we pass the correct address to printf.

Now let us see what happens when this file is linked into an executable. The section table now looks something like Listing 2.

The first thing you will notice is a lot more sections than were in the simple .o file. Much of this because this file requires the ELF shared library libc.so.1.

At this point I should mention the mechanics of what happens when you run an ELF program. The kernel looks through the binary and loads it into the user's virtual memory. If the application is linked to a shared library, the application will also contain the name of the dynamic linker that should be used. The kernel then transfers control to the dynamic linker, not to the application. The dynamic loader is responsibile for first initializing itself, loading the shared libraries into memory, resolving all remaining relocations, and then transferring control to the application.

Going back to our executable, the .interp section simply contains an ASCII string that is the name of the dynamic loader. Currently this will always be /lib/elf/ld-linux.so.1 (the dynamic loader itself is also an ELF shared library).

Next you will notice 3 sections, called .hash, .dynsym, and .dynstr. This is a minimal symbol table used by the dynamic linker when performing relocations. You will notice that these sections are mapped into virtual memory (the virtual address field is non-zero). At the very end of the image are the regular symbol and string tables, and these are not mapped into virtual memory by the loader. The .hash section is just a hash table that is used so that we can quickly locate a given symbol in the .dynsym section, thereby avoiding a linear search of the symbol table. A given symbol can typically be located in one or two tries through the use of the hash table.

The next section I want to mention is the .plt section. This contains the jump table that is used when we call functions in the shared library. By default the .plt entries are all initialized by the linker not to point to the correct target functions, but instead to point to the dynamic loader itself. Thus, the first time you call any given function, the dynamic loader looks up the function and fixes the target of the .plt so that the next time this .plt slot is used we call the correct function. After making this change, the dynamic loader calls the function itself.

This feature is known as lazy symbol binding. The idea is that if you have lots of shared libraries, it could take the dynamic loader lots of time to look up all of the functions to initialize all of the .plt slots, so it would be preferable to defer binding addresses to the functions until we actually need them. This turns out to be a big win if you only end up using a small fraction of the functions in a shared library. It is possible to instruct the dynamic loader to bind addresses to all of the .plt slots before transferring control to the application--this is done by setting the environment variable `LD_BIND_NOW=1` before running the program. This turns out to be useful in some cases when you are debugging a program, for example. Also, I should point out that the .plt is in read-only memory. Thus the addresses used for the target of the jump are actually stored in the .got section. The .got also contains a set of pointers for all of the global variables that are used within a program that come from a shared library.

The .dynamic section contains some shorthand notes used by the dynamic loader. You will notice that the section table is not itself loaded into virtual memory, and in fact it would not be good for performance for the dynamic loader to have to try to parse it to figure out what needs to be done. The .dynamic section is essentially just a distilled version of the section header table that contains just what is needed for the dynamic loader to do its job.

You will notice that since the section header table is not loaded into memory, neither the kernel nor the dynamic loader will be able to use that table when loading files into memory. A shorthand table of program headers is added to provide a distilled version of the section table containing just the information required to load a file into memory. For the above file it looks something like:

```
largo% readelf -l hello
Elf file is Executable
Entry point 0x8000400
There are 5 program headers, starting at offset 34:
PHDR         0x00034 0x08000034 0x000a0 0x000a0 R E
Interp       0x000d4 0x080000d4 0x00017 0x00017 R
Requesting program interpreter
[/lib/elf/ld-linux.so.1]
Load         0x00000 0x08000000 0x00515 0x00515 R E
Load         0x00518 0x08001518 0x000cc 0x000d4 RW
Dynamic      0x0054c 0x0800154c 0x00098 0x00098 RW
Shared library: [libc.so.4] 1
```

As you can see, the program header contains a pointer to the name of the dynamic loader, instructions on what portions of the file are to be loaded into virtual memory (and the virtual addresses they should be loaded to), the permissions of the segments of memory, and finally a pointer to the .dynamic section that the dynamic loader will need. Note that the list of required shared libraries is stored in the .dynamic section.

I will not pick apart an ELF shared library for you here--libraries look quite similar to ELF executables. If you are interested, you can get the readelf utility and pick apart your own libraries.

At the start of this article, I said one reason we were switching to ELF was that it was easier to build shared libraries with ELF. I will now demonstrate how. Consider two files:

```
largo% cat hello1.c
main()
{
        greet();
}
largo% cat english.c
greet()
{
        printf("Hello World
");
}
```

The idea is that we want to build a shared library from english.c, and link hello1 against it. The commands to generate the shared library are:

```
largo% gcc-elf -fPIC -c english.c
largo% gcc-elf -shared -o libenglish.so english.o
```

That's all there is to it. Now we compile and link the hello1 program:

```
largo% gcc-elf -c hello1.c
largo% gcc-elf -o hello1 hello1.o -L. -lenglish
```

And finally we can run the program. Normally the dynamic loader only looks in certain locations for shared libraries, and the current directory is not one of the places it normally looks. Thus to run the program, you can use a command like:

```
largo% LD_LIBRARY_PATH=. ./hello1
Hello World
```

The environment variable `LD_LIBRARY_PATH` tells the dynamic loader to look in additional places for the shared libraries (this feature is disabled for setuid programs for security reasons).

To avoid having to specify `LD_LIBRARY_PATH`, you have several options. You could copy your shared library to /lib/elf, but you can also link your program in the following way:

```
largo% gcc-elf -o hello1 hello1.o /home/joe/libenglish.so
largo% ./hello1
Hello World
```

To build more complicated shared libraries, the procedure is not really that much different. Everything that you want to put into the shared library should be compiled with `-fPIC`; when you have compiled everything, you just link it all together with the `gcc -shared` command.

The procedure is so much simpler mainly because we bind addresses to functions at runtime. With a.out libraries, the addresses are bound at link time. This means that lots of special care must be taken to ensure that the .plt and .got have sufficient room for future expansion and that we keep the variables at the same addresses from one version of the library to the next. The tools for building a.out libraries help ensure all of this, but it makes the build procedure much more complicated.

ELF offers one further feature that is not easily available with a.out. The **dlopen()** function can be used to dynamically load a shared library into the user's memory, and you are then able to call the dynamic loader to find symbols within this shared library--in other words, you can call functions that are defined in these modules. In addition, the dynamic loader is used to resolve any undefined symbols within the module itself.

This may be easiest to explain with an example. Given the following source file:

```
#include <dlfcn.h>
main(int argc, char * argv[])
{
   void (*greeting)();
   void * module;
   if( argc > 2 ) exit(0);
   module = dlopen(argv[1], RTLD_LAZY);
  if(!module) exit(0);
   greeting = dlsym(module, "greet");
   if(greeting) {
     (*greeting)();
   }
  dlclose(module);
 }
```

you can compile, link, and run it (using the shared library english.so which was built earlier):

```
largo% gcc-elf -c hello2.c
largo% gcc-elf -o hello2 hello2.o -ldl
largo% ./hello2 ./libenglish.so
Hello World
```

To expand this example a little bit, you could generate other modules with greetings in other languages. Thus in theory, one could add multi-lingual support for some application merely by supplying a set of shared libraries that contain the language-specific portions of the application. In the above example, I showed how you can locate the address of a function within a shared library. But the **dlsym()** function will also return the address of data variables, so you could just as easily retrieve the address of a text string from the shared library.

As I prepare to close, I should mention some options to readelf which I have not demonstrated. `readelf -s` dumps the symbol tables and `readelf -f` dumps the .dynamic section.

Finally, I should mention something about the timetable. When we first got ELF to a point where it was usable (last September), we decided to spend a relatively long period of time testing it and shaking out all of the problems. Back then I felt that roughly 4-to-6 months would allow people to test it thoroughly, plus we wanted to give an opportunity for certain applications to be adapted for ELF (the most recent versions of insmod and Wine now support ELF, for example). As I write this, no firm date has been set for a public release, but it is possible that ELF will be public by the time you read this.

In these articles I have attempted to give you a guided introduction to the ELF file format. A lot of the material I have covered is not of much practical value to most users (unless you want to hack the linker), but my experience is that there are a lot of people who are curious about how it all works, and I hope that I have provided enough information to satisfy most people.

For more information about the ELF file format, you can obtain the ELF specifications from a number of sources--you can try ftp.intel.com in pub/tis/elf11g.zip. The specifications are also available in a printed format. See *SYSTEM V Application Binary Interface* (ISBN 0-13-100439-5) and *SYSTEM V Application Binary Interface*, Intel386 Architecture Processor Supplement (ISBN 0-13-104670-5).

Eric Youngdale has worked with Linux for over three years, and has been active in kernel development. He developed the current a.out Linux shared libraries before developing much of the new ELF support. He can be reached as eric@aib.com.

Google

Web Images Groups News Froogle New! **more »**

library function version product information | Search | Advanced Search | Preferences

Web Results **1** - **10** of about **463,000** for **library function version product information**. (**0.33** seconds)

Intel® Software **Products**
... versions of Intel® Software Development **Products** are available and will cease to **function** at the ... Integrated Performance Primitives (IPP) **library** is composed ...
www.intel.com/software/**product**s/global/eval.htm - 69k - May 3, 2004 - Cached - Similar pages

Vector Math **Library** (VML) Notes
... VML is an integral part of the Intel® Math Kernel **Library** and the VML terminology is used ... For many **functions**, using the LA **version** improves performance at ...
www.intel.com/software/**product**s/ mkl/docs/vmlnotes/vmlnotes.htm - 8k - Cached - Similar pages
[More results from www.intel.com]

Guide and Reference
... Changes for ESSL **Version** 3; Future Migration. ... Usability; The Variety of Mathematical **Functions**; ESSL--Processing ... an ESSL Subroutine; Which ESSL **Library** Do You Want ...
csit1cwe.fsu.edu/extra_link/essl/essl002.html - 55k - Cached - Similar pages

Amazon.com: Books: C Programming Language (2nd Edition)
... addition to, or instead of, the **product** on this ... developers of C, this new **version** helps readers ... exercises indicate how some standard **library functions** might be ...
www.amazon.com/exec/obidos/tg/ detail/-/0131103628?v=glance - 86k - May 4, 2004 - Cached - Similar pages

MySQL® Database Server
... Using the embedded database **library** (libmysqld), you can ... **Version** 4.1 includes support for a subset of ... procedures allow you to create **functions** and subroutines ...
www.mysql.com/**product**s/mysql/ - 17k - May 3, 2004 - Cached - Similar pages

Global Optimization: **Product Information**
... **Version** 4.3 provides dramatic improvements in speed for ... A **library** of speed-optimized common univariate **functions** is ... InterchangeMethodMin is a **function** for 0-1 ...
www.wolfram.com/**product**s/applications/globalopt/ - 24k - May 3, 2004 - Cached - Similar pages

Visual Numerics - Developers of IMSL and PV-WAVE
... To find other company and **product information**, please click here. ... MB PDF) IMSL C Numerical **Library Version** 5.5 (CNL) Functionality CNL **Function** Catalog (0.3 ...
www.vni.com/**product**s/imsl/documentation/ - Similar pages

Guide and Reference
... 3 Release 1.1; Changes for ESSL **Version** 3; Future ... Usability; The Variety of Mathematical **Functions**; ESSL--Processing ... an ESSL Subroutine; Which ESSL **Library** Do You ...
webdocs.caspur.it/ibm/web/essl-3.2/essl002.html - 56k - Cached - Similar pages

Installer **Function** Reference [Windows Installer]
... must call CoInitialize to initialize the COM **library** with a ... This **function** does not affect the registered source list ... Available with Windows Installer **version** 3.0 ...
msdn.microsoft.com/**library**/en-us/msi/setup/ installer_**function**_reference.asp - 27k -

Sponsored Links

Product information
Outsourced data service - **product** attributes for over 60 categories
www.ActiveDecisions.com

The Source of **Information**
Your Great Source for "How To" Information Products on the Net-aff
www.cbmall.com

See your message here...

Google

Web Images Groups News Froogle New! more »

library version information linked object | Search | Advanced Search Preferences

Web Results **1** - **10** of about **267,000** for **library version information linked object**. (**0.32** seconds)

The Rexx language
... Rexx GUI Front-End and the LesteclO Rexx I/O **Library**, **version** 2.1, are ... to use the
SideKick plugin and works with prefinal **version** of jedit. ... Related **information**. ...
www2.hursley.ibm.com/rexx/ - 16k - Cached - Similar pages

Free Pascal Programmers' manual
... divide Automatic alignment Smart **linking** Inline routines ... 11.4 Using a pascal **library**
from aC ... 12.4 Inserting **version information** 12.5 Inserting an application ...
www.freepascal.org/docs-html/prog/prog.html - 41k - Cached - Similar pages

Libtool
... avoid versioning (see section 6. **Library** interface **versions** ... libraries and modules,
ie no **version information** is stored ... or if the program is **linked** with `-static ...
www.delorie.com/gnu/docs/libtool/libtool_17.html - 13k - May 4, 2004 - Cached - Similar pages

zlib Home Site
... zlibCompileFlags() to provide compilation **information**. More supported architectures ... available)
zlib Pascal port (**version** 1.1.2 ... part of the standard **library** as of ...
www.zlib.org/ - 34k - Cached - Similar pages

Open Source Initiative OSI - The LGPL: Licensing
... 2.1 of the License, or (at your option) any later **version**. ... General Public License
along with this **library**; if not ... Also add **information** on how to contact you by ...
www.opensource.org/licenses/lgpl-license.php - 34k - Cached - Similar pages

The World Wide Web Virtual **Library**: Logic Programming
* Virtual **Library** * Computing * Languages * Prolog * AI ... Email info@amzi.com for **information**. ... ProWindows
3 is a commercial **version** of XPCE for Quintus Prolog. ...
archive.comlab.ox.ac.uk/logic-prog.html - 50k - Cached - Similar pages

PNG (Portable Network Graphics) Home Site
... PNG Programming **Information**: ... to PNG known as MNG is officially complete (**version**
1.0 of ... a number of applications available and a free reference **library**, too. ...
www.libpng.org/pub/png/ - 26k - Cached - Similar pages

Stroustrup: C++
... An ASCII **version** of the slides I used for my keynote at ... The learn.c-c++ newsgroup
FAQ presents much **information** of use for C ... STL (Standard Template **Library**) FAQ ...
www.research.att.com/~bs/C++.html - 13k - May 3, 2004 - Cached - Similar pages

Linkers -- Indiana University
... be position dependent (see man ld for more **information**. ... To use the IBM-optimized BLAS
library on the ... lapack -llapack (or -llapack90 for the Fortran90 **version**). ...
www.indiana.edu/~rac/hpc/pl/linkers.html - 11k - Cached - Similar pages

Object Support **Library Version** History
... References. Technote 1083: Weak-**Linking** to a Code Fragment Manager-based Shared
Library. ... Acrobat **version** of this Note (60K). ... Get **information** on Apple products. ...

Sponsored Links

Library Software
Library and info mgmt. solutions
for libraries, archives, & museums
www.library.geac.com

Learn Microsoft Access
Build MS Access Databases Like a
Pro. Step-by-Step Videos. Free Demo
http://www.599cd.com/Access/Free

See your message here...

Google

Web Images Groups News Froogle New! **more »**

function name version Search Advanced Search Preferences

Web Results **1** - **10** of about **2,830,000** for function name version. (**1.53** seconds)

XML Path Language (XPath)
... The English **version** of this specification is the only normative **version**.
However ... nodes. **Function**: string **name**(node-set?). The ...
www.w3.org/TR/xpath - 101k - Cached - Similar pages

XSL Transformations (XSLT)
... template match="/">xsl:choose> <xsl:when test="system-property('xsl:**version**') &lt;
1.1 ... if the expression calls a **function** with an unprefixed **name** that is ...
www.w3.org/TR/xslt - 101k - Cached - Similar pages
[More results from www.w3.org]

WINDOWS EXECUTABLE
File Format: Unrecognized - View as HTML
... Operating System **Version**: 1.00. Image **Version**: 0.00. Subsystem **Version**: 4.00.
Reserved1: 00000000. ... kernel32.dll. Ordinal **Function Name**. 0000 GetCurrentThreadId. ...
www.skype.com/go/getskype - Similar pages

WINDOWS EXECUTABLE
File Format: Unrecognized - View as HTML
... 0134 StringFromGUID2. 000f CoCreateGuid. WS2_32.dll. Ordinal **Function Name**. **VERSION**.dll.
Ordinal **Function Name**. 0003 GetFileVersionInfoW. 0002 GetFileVersionInfoSizeW. ...
www.microsoft.com/windows2000/technologies/ fileandprint/print/download.asp - Similar pages

Function Names for Web Applications (German - English)
... Generate. This **function name** should be limited to the development environment.
Hilfe. ... Copy. To move a second **version** of an object to another location. Löschen. ...
www.sapdesignguild.org/resources/ References/**function_names**_de.htm - 32k - Cached - Similar pages

Function Names for Web Applications (English - German)
... Kopieren. To move a second **version** of an object to another location. Create. ... Erzeugen.
This **function name** should be limited to the development environment. Help. ...
www.sapdesignguild.org/resources/ References/**function_names**_ed.htm - 36k - Cached - Similar pages
[More results from www.sapdesignguild.org]

Can I use variable in **function name**?
... num = 4; eval("Message" . $num . "();. Read Formatted **Version**.
Can I use variable in **function name**? itsource. I get error message ...
forums.devshed.com/archive/t-48766 - 20k - Cached - Similar pages

WINDOWS EXECUTABLE
File Format: Unrecognized - View as HTML
... **Version**: 0.00. Base: 00000001. Number of **Functions**: 00000000. Number of **Names**:
00000000. - No exported **functions**. Import Table. USER32.dll. Ordinal **Function Name**. ...
www.winzip.de/downloadb.htm - Similar pages

PHP Bugs: #24286: using array from class as **function name**
... go to bug id or search bugs for. Bug #24286, using array from class as **function**
name. ... **Version**: 5CVS-2003-06-22 (dev), OS: RedHat 9.0. Votes: 4, Avg. ...
bugs.php.net/bug.php?id=24286 - 11k - Cached - Similar pages

EXSLT - func:**function** - **Version** 2
Version: 2 Previous **Version**: func.**function**.1.html Status: revised Element Package:

PORTAL
US Patent & Trademark Office

Subscribe (Full Service) Register (Limited Service, Free) Login

Search: [✓ The ACM Digital Library] The Guide

+abstract:function +abstract:name +abstract:version SEARCH

THE ACM DIGITAL LIBRARY

Feedback Report a problem Satisfaction survey

Terms used function name version

Found **7** of **12,846 searched out of 12,846.**

Sort results by: relevance
Display results: expanded form

Save results to a Binder
Search Tips
Open results in a new window

Try an Advanced Search
Try this search in The ACM Guide

Results 1 - 7 of 7

Relevance scale

1 Namespaces: APL/W vs. APL2
Rexford H. Swain
June 1995 **ACM SIGAPL APL Quote Quad , Proceedings of the international conference on Applied programming languages**, Volume 25 Issue 4
Full text available: pdf(1.08 MB) Additional Information: full citation, abstract, references, citings, index terms

This paper describes and contrasts the implementation of namespaces in two popular combinations of APL dialects and platforms: IBM's APL2 (version 2, release 2) running under VM/CMS, and Dyadic's Dyalog APL/W (version 7.0) running under Microsoft Windows.In a traditional APL workspace, localization is the only mechanism available to isolate identifier names and values, and it is extremely potent. While a function that localizes a given name is executing, it is impossible to refer ...

Keywords: localization, namespace, package

2 The CL-PVM package
Liwei Li, Paul S. Wang
December 1995 **ACM SIGSAM Bulletin**, Volume 29 Issue 3-4
Full text available: pdf(269.90 KB) Additional Information: full citation, abstract, citings, index terms

Parallel Virtual Machine (PVM) is a software package that integrates a heterogeneous network of computers to form a single parallel/concurrent computing facility [2]. PVM consists of two parts: a run-time server and a set of library functions. A user sets up a *hostfile* that lists the names of the hosts constituting the *parallel virtual machine (pvm).* A PVM server runs on each host to help manage the *pvm.* Hosts can be added and deleted from the *pvm* dynamically. ...

3 An example of process description in HFSP
Takuya Katayama, Masato Suzuki
October 1990 **Proceedings of the 5th international software process workshop on Experience with software process models**
Full text available: pdf(165.82 KB) Additional Information: full citation, abstract, references, index terms

HFSP considers software activities, in the first approximation, as mathematical functions which map their input objects to output objects and define them though hierarchical functional definition. Activity A with input x1, ..., xn and output y1, ..., ym is denoted by A (x1 ...

4 Computing with text-graphic forms
Fred H. Lakin

August 1980 **Proceedings of the 1980 ACM conference on LISP and functional programming**
Full text available: pdf(398.65 KB) Additional Information: full citation, abstract, references, citings, index terms

Computing with text-graphic forms occurs when text-graphic patterns are used to direct the processing of other text-graphic patterns. The PAM graphics system was designed for just this kind of computation; PAM stands for PAttern Manipulating—PAM is a generalization of LISP (McCarthy [0]) from computing with (textual) symbolic expressions to computing with text-graphic forms. Like LISP, PAM acheives processing power by provi ...

5 Review of Isomorphisms of Types:: from λ-calculus to information retrieval and language design
Roberto Di Cosmo
December 1997 **ACM SIGACT News**, Volume 28 Issue 4
Full text available: pdf(254.19 KB) Additional Information: full citation, abstract

The λ-calculus is an elegant computational model used extensively in the formal semantics of programming languages. It was first described by Alonzo Church in 1932, as a foundation of mathematics based not on *sets,* but directly on *functions.* λ-calculus was, in part, the inspiration for the programming language Lisp, and its typed versions have become the foundation for a family of modern type-safe languages that includes Haskell and ML.

In its ori ...

6 On the orthogonality of assignments and procedures in Algol
Stephen Weeks, Matthias Felleisen
March 1993 **Proceedings of the 20th ACM SIGPLAN-SIGACT symposium on Principles of programming languages**
Full text available: pdf(1.27 MB) Additional Information: full citation, abstract, references, citings, index terms

According to folklore, Algol is an "orthogonal" extension of a simple imperative programming language with a call-by-name functional language. The former contains assignments, branching constructs, and compound statements; the latter is based on the typed &lgr;-calculus. In an attempt to formalize the claim of "orthogonality", we define a simple version of Algol and an extended &lgr;-calculus. The calculus includes the full &bgr;-rule and rules for t ...

7 Update and retrieval in a relational database through a universal schema interface
Volkert Brosda, Gottfried Vossen
October 1988 **ACM Transactions on Database Systems (TODS)**, Volume 13 Issue 4
Full text available: pdf(3.16 MB) Additional Information: full citation, abstract, references, citings, index terms, review

A database system that is based on the universal relation (UR) model aims at freeing its users from specifying access paths on both the physical and on the logical levels. All information about the logical structure of the database (i.e., its conceptual scheme) is hidden from users; they need only to know the attribute names, which now carry all the semantics of the database. Previous work on UR interfaces has concentrated on the design and implementation of query languages that ...

Results 1 - 7 of 7

Google

Web Images Groups News Froogle New! more »

library version executable Search Advanced Search Preferences

Web Results **1** - **10** of about **165,000** for library version executable. (**0.40** seconds)

GNU **Library** General Public License - GNU Project - Free Software ...
... Such a contradiction means you cannot use both them and the **Library** together in an **executable** that you ... If the **Library** specifies a **version** number of ...
www.gnu.org/copyleft/lgpl.html - 30k - Cached - Similar pages

GNU Lesser General Public License - GNU Project - Free Software ...
... user's computer system, rather than copying **library** functions into the **executable**, and (2) will operate properly with a modified **version** of the **library**, if the ...
www.gnu.org/copyleft/lesser.html - 31k - Cached - Similar pages
[More results from www.gnu.org]

EEL: An **Executable** Editing **Library**
... an object, **library**, or **executable** file. A tool opens an **executable**, examines and modifies its contents, and writes an edited **version**. An **executable** primarily ...
www.cs.wisc.edu/~larus/eel.html - 7k - Cached - Similar pages

Mats Weber's Ada Component **Library, version** 2.0
... does not however invalidate any other reasons why the **executable** file might be ... I will correct it and release a new **version** of the whole **library**, instead of ...
lglwww.epfl.ch/Team/MW/mw_components.html - 19k - Cached - Similar pages

HTML Tidy Project Page
... Classic. All development is now done on the **library version** of Tidy, otherwise known as TidyLib. The ... Support. **Executable** binaries. If ...
tidy.sourceforge.net/ - 29k - Cached - Similar pages

Business Process Execution Language for Web Services **Version** 1.1
... microsoft.com/**library**/default.asp?url=/**library**/en-us ... of business protocol description and **executable** business process ... In the 1.1 **version** of the specification we ...
www-106.ibm.com/developerworks/**library**/ws-bpel/ - 101k - May 3, 2004 - Cached - Similar pages

Set and retrieve the **executable version** - Real's PowerBuilder How ...
... **executable** : [local external function declaration] FUNCTION ulong GetFileVersionInfoSizeA & (REF string lpFilename, REF ulong lpdwHandle) & **LIBRARY** "**version**. ...
www.rgagnon.com/pbdetails/pb-0120.html - 5k - Cached - Similar pages

Free Xtal_GX distribution page
... Linux 2.0.29 (g77) elf shared **library version** 1 Meg **version** 1. DOS **executable** for PCs 0.69Meg. The following auxilliary files are essential. ...
journals.iucr.org/iucr-top/cif/software/xtal/gx/ - Similar pages

RCS **Library Version** Functions
... Using What to determine the **Version**. The SCCS what command can be used to determine the **version** of the **library** linked into an **executable**. ...
www.isd.mel.nist.gov/projects/rcslib/rcsvers.html - 3k - May 4, 2004 - Cached - Similar pages

The bzip2 and libbzip2 home page
... Try these if you can't get a 1.0.2 **executable** for your ... **Version** 1.0.0 (and 1.0.1; these are functionally identical ... The compression **library** underlying 1.0.0/1.0.1. ...
sources.redhat.com/bzip2/ - 16k - Cached - Similar pages